Journal of Theoretical and Applied Information Technology
15th July 2016. Vol.89. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SCALABLE TRANSACTIONS USING MONGODB CLOUD DATA STORE

[1] LALITA CHAPLE, [2]DR. SWATI AHIRRAO
[1,2]Computer Science Symbiosis Institute of Technology (SIT),
Symbiosis International University (SIU), Lavale, Pune, India,
E-mail: [1]lalita.chaple@sitpune.edu.in , [2]swatia@sitpune.edu.in

ABSTRACT

Cloud computing exists as a rising technology that provides the functionality of available and highly scalable web applications at extremely low cost. Building a scalable, available and consistent cloud data store is the challenge now a day's. Scalability is improved by using primary technique that is data partitioning. Classical, Partitioning strategies are not able to track the patterns of the web applications. This paper implements workload-driven approach rest on data access patterns in MongoDB for improving scalability. As observed from results it generates less number of distributed transactions. Similarly, implementation and validation of scalable workload-driven partitioning scheme are accomplished via experimentation over cloud data store (MongoDB). An experimental outcome of the partitioning strategy is managed by utilizing the industry benchmark TPC-C.

Keywords: *Scalability, Oltp, Data Partitioning, Partitioning Scheme, Workload-Driven.*

1. INTRODUCTION

Cloud computing has been derived as an extensive model for introducing web scale applications in huge computing infrastructures. The major enabling aspects of cloud comprise of on demand services, the elasticity of resources, the notion of limitless resources and enormous scalability. Probably the most primary use of the cloud is for a web hosting the vast number of internet applications, scalable data management techniques that force these applications, are a significant component into the cloud [4]. Constructing scalable and uniform data management has been the aim of database investigator for the last few years. Many applications on the web are deployed with its rising fame. They have suffered the problem of serving customers in thousands. Hence, scalability of browsing internet purposes has turned out to be the primary issue. These recent internet applications develop the massive quantity of data. The database management system plays vital role in managing a huge quantity of data.

As the DBMS is beneficial to preserve uniform and affordable performance, it has to scale out at low price commodity hardware. Conventionally, Relational databases might not be scaled out at low price commodity servers. This gives a start to the NoSQL data stores [16]. NoSQL data stores include services such as availability, elasticity and scalability. The primary approach to achieve scalability is partitioning the data. In an e-commerce application, when a customer wants to buy an item then request is processed by the warehouses. Whenever the warehouses on to the same partition are out of stock, an order is fulfilled through the warehouses on another partition [14]. Here, the pattern is formed that is which warehouse is most probable supplier warehouse for the processing warehouse. This behavior is tracked and the pattern is recognized. This pattern is called as Data Access Patterns [14]. Static partitioning techniques are techniques, in which partitions are formed by collocating the associated data items on one partition. Once partitions are formed, they do not change [4]. These partitioning techniques are referred as **static partitioning**. In dynamic partitioning techniques, partitions are changed dynamically but it costs overhead.

In scalable workload-driven partitioning [14], analyzing a transaction logs and monitoring data access patterns that is data changes periodically is introduced in MongoDB. Traditional Partitioning mechanisms such as range, hash are simple to implement but generates large number of distributed transactions and does not track data

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

access patterns. We provide implementation of scalable workload-driven partitioning [14] in MongoDB. Scalable workload-driven partitioning is especially implemented for OLTP internet applications. This partitioning scheme allows execution of fewer amounts of shared transactions during transactions. The main contribution of the paper is structured as follows:

- Design of workload-driven partitioning in MongoDB is introduced. It consistently handles the load with all partitions. The utilization of this workload-driven partitioning to restrict the transaction to one partition is exhibited.
- Implementation of workload-driven partitioning in MongoDB.
- The practical demonstration of the scalable workload-driven partitioning in a cloud data stores (MongoDB) is presented. It is evaluated using industry benchmark TPC-C.

Further paper is arranged as follows:

The survey of different partitioning techniques is performed in Section2. Section 3 discusses issues in existing system. The proposed work is presented in section 4. Section 5 consists of implementation details and Conclusion in Section 6.

2. RELATED WORK

Researchers have implemented various systems and partitioning techniques to upgrade the scalability of the transactions for web applications. In this paper, partitioning techniques which improve the scalability are studied.

2.1 Partitioning Algorithms

2.1.1 Schema level partitioning

In schema level partitioning [4], partitioning depends on partitioning key and all associated items are placed on single partitions to avoid distributed transactions. Schema level partitioning is a static partitioning once the partitions are formed, it remains same. Schema level partitioning, limit transactional access to only one database partition. Schema level partitioning has used the technique that accesses large number database schemas but transactions access only less number of associated data items which is spread over different tables.

2.1.2 Graph partitioning

In Graph partitioning [5], each node in the graph represents the tuple. All related data items related to that node are linked to another node via edge, among two nodes. In the first phase, a graph is created with the node according to a tuple and edges within nodes are accessed by using the same transaction, later apply graph partitioner to separate the graph within k-balanced partitions to restrict the variety of cross-partition transactions [5]. The second phase makes use of machine learning methodologies to find a predicate-based explanation of segregating scheme.

2.2 Cloud Data Stores

The partitioning techniques for different cloud data stores [1] and partition the cloud data store based on the partitioning and non-partitioning is discussed. The main aim of the author is a scalable database. Scalable workload-driven partitioning [14] which generates partition based on data access patterns of web applications. This is neither static nor dynamic. It lies between static and dynamic partitioning. Dynamic group formation can be completed by using association mining. Two approaches for association mining that is Apriori and FP-Growth.

ElasTraS [4] comes into picture from the scalable Key-Value stores to decrease distributed transactions and eliminate scalability bottlenecks, as all know that partitioning the database is used for providing scalability. ElasTraS are based upon schema level partitioning scheme of the database to assist capability, despite the fact that transactions are restricted to single partitions [4]. Discover design concepts which are utilized in designing scalable systems concurrently assuming transactional guarantees. Elastras is appropriate for internet applications which access patterns are static. Elastras only focus on scalability, not

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

availability so no need to concentrate on replication.

TAVPD [2] presents transaction-aware partitioning. It uses vertical partitioning for improving scalability. The main aim of author's [2] is to decrease response time and optimized cost of processing. The first approach is mini transactions, transactions are splits into sub-transactions. The second technique is partitioning the data that is a primary technique used for upgrading the performance of database that is scalability, implementing selective consistency with a classification of the data on their consistency index.

The automating entity group which is a first paper who discuss the process of automatic creating entity groups [6]. Earlier days, transactions were done in the same entity group was guarantee full ACID properties but the problem with different entity groups. Author automatically creates entity-group and customer has to deliver feedback for automatically creating entity group. The author uses an edge-coverage algorithm for automatically creating entity-group.

The Deuteronomy [8] discusses the effective and scalable transactions techniques by separating the data and transaction component. The idea of this paper consists of two separate components, 1) data component doesn't know about the transaction process and 2) transaction component doesn't know about physical address of the data. Authors discuss the optimization methodologies which aid to upgrade the throughput and increase the performance of the system.

The Schism [5] has graph partitioning strategies which increase the scalability. It uses partitioning for enhancing scalability and replication for availability approach for distributed databases. Partitions are organized based on the graph which contains nodes and edges. Schism models the workload as a graph and enforces k-way min-cut graph partitioning algorithm to lessen the effect of distributed transactions and increases the throughput. It uses static partitioning technique that is partitions are initiated only once and remain forever.

Megastore [7] combines the scalability of a NoSQL data store with regard to a traditional RDBMS and offers the strong consistency and high availability. Megastore gives full ACID properties inside the partitions. This system partitions the data into multiple entity groups i.e. the set of correlated data items which independently replicated over the set of servers and each update is replicated synchronously across the identical partitions with acceptable latency. The transaction which requires the strong consistent view of the database to fulfill its execution restricts its execution to a single entity group.

Cloud TPS [9] for a scalable transaction in the web applications which has followed strict ACID properties. It splits transaction manager within multiple Local Transaction Manager (LTMs) to support scalable transactions. The items are assigned to LTM applying consistent hashing mechanism to accomplish high scalability. Cloud TPS contains static partitioning technique to enhance scalability and suitable for those applications, whose access pattern act as static.

3. ISSUES

Sharding the database is the primary techniques to scale out a database to different nodes. Normally, a database is divided by splitting the individual tables within the database. Common partitioning techniques used are range partitioning or hash partitioning. Range partitioning contains dividing the tables into non-overlapping ranges of their keys and then mapping the ranges to a set of nodes. In hash partitioning, the hash keys are utilize to assign rows across various nodes. These partitioning methodologies are simple to implement, but the main limitation of such techniques is that they result in the need for large numbers of distributed transactions to access data partitioned across various servers. As a result, the scalability of such systems is limited due to the costs of distributed transactions.

Therefore, the challenge is to partition the database such that most accesses are limited to a single partition. Existing techniques are worked well when data access patterns are static but when it dynamically changes it costs overhead. Considering all these issues we are designed and implemented a system in MongoDB to overcome these limitations. It offers the less quantity of distributed transactions, less response time and higher throughput.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4. PROPOSED WORK

4.1 Design of Scalable Workload-Driven Partitioning in MongoDB

Now a day's e-commerce application plays an important role, so there are enormous numbers of customers who place an order for different items. So pattern is formed between these that is which warehouse places an order, if current warehouse is out of stock, then supplier warehouse processes that item. This process which is tracked is called data access patterns [14] as shown in Figure 1. Those patterns are analyzed and transaction logs are monitored, So that distributed transactions are avoided and scalability is accomplished. For example, if customer from warehouse in Pune wish to place an order but the item is out of stock then it is fulfilled by another warehouse in Mumbai. These warehouses have to be located on same partition to minimize distributed transactions. Likewise these patterns are changed dynamically by analyzing transactions logs and monitoring access patterns.

4.2 Data Partitioning Technique

In this partitioning, an extensive analysis is done to discover the top load distribution. All feasible combinations of partition are identified as shown in Figure 2.

The total association and load are estimated for all feasible combinations of partition. The Heuristic Search mechanism is utilized to discover advance solutions. The particular combinations are obtained applying mutation in the genetics algorithm. Mutation is the mechanism that is utilized in genetic algorithms for originating variation. Mutation aids in obtaining better combinations [14]. In mutation, the result may alter totally from the previous result. Therefore, a genetic algorithm can occur to a superior solution by utilizing mutation. The partition with feasible association and load are chosen and given a higher throughput.

4.3 Load

Load is estimated by using the warehouse data (transaction data) that is warehouse included in the database. The transactions are executed and the warehouses are found out while executing transactions. The static distribution of those warehouses to form the partition is estimated [14]. If there are six warehouses w1, w2, w3, w4, w5, w6 then distribute them as w1 and w2 on one partition, w3 and w4 on one partition and w5 and w6 on another partition. All transaction data regarding those warehouses should be accessed while executing transactions. After the static distribution of warehouses, find out all the combination of warehouses according to mutation algorithm. Estimate the average load by summation of all partition load divided by number of partitions involve. Lastly find the load on that particular partition that is number of transactions executing on that partition from average load.

4.4 Association

Association is estimated by finding out the number of local transaction executed and number of remote transactions (distributed transaction) executed on that partition. Number of transaction means the transaction is performed by home warehouse that is customer of warehouse w1 is placed an order and an order is processed by warehouse w1. If the order is processed by another warehouse that is customer is related to w1 and order is processed by warehouse w2 that is remote warehouse. The association of number of transaction and distributed transaction executed on the partition should be estimated.

4.5 Scalable Workload-Driven Partitioning Algorithm

Scalable workload-driven partitioning algorithm [14] takes input as number of warehouses, transaction data and gives output as optimized partitions. Firstly the warehouses have to be distributed statically. For example, if there are 4 warehouses as w1, w2, w3, w4 then distribute them statically that is w1, w2 is combination on one partition and w3, w4 is combination on another partition. The number of combinations of all warehouses by using mutation algorithm should be found out after that. Estimate the load distribution [14] of all possible combination using standard deviation to find out the load on each partition. The Rank the combination as higher rank value to the higher standard deviation and lower rank value to the lower standard deviation. Association [14] of combinations is estimated by executing the local transactions and number of distributed transaction on that particular combination. The ranking of association depends on association, higher rank value to the lower association and lower rank value to the higher association. Perform the addition of load rank and association rank. Arrange these

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

combinations in ascending order. Select the top combinations on the basis of top ranks.

Algorithm 1: Scalable Workload-Driven Partitioning Algorithm

Input: Number of warehouses, Transaction data
Output: Optimized load and association for the partition.

Step1: Static Distribution of warehouses.
Step2: Find all the possible combination of partition.
Step3: Calculate the load distribution for all possible combination using standard deviation.
Step4: Sort the load distribution in ascending order.
Step5: Calculate the association by executing the number of transactions and distributed transactions on that combination.
Step6: Sort the association in descending order.
Step7: Summation of both load rank and association rank.
Step8: Sort the rank value in ascending order.
Step9: Select top two combinations.
Step10: Select the top combinations which have optimized load and association.

After algorithm is running, it gives the optimized load and association as an output. Then the last solution which is generated from scalable workload driven partitioning is utilize to populate the data. Mentioned algorithm, scans the transaction log and builds all various combinations of the feasible partitions and estimate the overall load and total association of that partition. Then, the rank values are allocated to each combination accordingly their load in ascending order and an association in descending order. Then, estimate the addition of both the load rank and association rank and generate the ultimate combination. Then high three partitions are chosen depending on the final rank and repeat steps two to nine for three times, to get more number of partitions. Once the combinations are generated, additionally choose top three partitions and execute the algorithm again from 2 to 9 for 3 times. Implementing this to form a more number of partitions to check whether the combinations that are generated are equal or not.

5. IMPLEMENTATION

This section includes implementation details of scalable workload-driven partitioning [14] in MongoDB. Various data models are supported by NoSQL data store. Challenge of scalable workload-driven partitioning is to execute the transaction with respect to response time having scalability and restrict the number of distributed transactions. Scalable workload-driven partitioning is being implemented by utilizing outstanding and generally used NoSQL data stores: MongoDB.

5.1 TPC-C Benchmark

TPC-C is the benchmark of industry which is utilized as simulating the e-business application workload [15]. It indicates the standard of measurement OLTP workload. The benchmark represents a wholesale provider with the geographically distributed warehouses and districts. Workload-Driven partitioning technique is enforced to a standard web application like TPC-C to exhibit its performance. Workload-Driven partitioning technique is evaluated by using a benchmark TPC-C. The Benchmark comprises of 5 transactions by analyzing business obligation of e-commerce applications:

5.1.1 New order

New order transaction [15] is the combination of read and writes transactions. It creates a new order for the customer and places an order according to the customer need.

5.1.2 Payment

Payment transaction [15] is also the combination of read and writes transaction. When a payment transaction is executed it updates the balance of customer.

5.1.3 Order status

Order status transaction [15] is read only transaction. It tracks a status of the customer that is customer's last order.

5.1.4 Delivery

Delivery transaction [15] is also a read and writes transaction. It consists of group of 10 new orders that is orders not yet delivered to the customer.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

5.1.5 Stock level

Stock level transaction [15] is ready only transaction. It decides the quantity of recently sold things which have stock below the threshold.

In reality, usually NEW ORDER transactions are 45%, PAYMENT transactions are 43% and ORDER STATUS, STOCK and DELIVERY transactions are 4%.

5.2 Conversion of TPC-C schema to MongoDB Collection

TPC-C was originally designed for internet applications with relational databases as backend, hence it is necessary to covert the data model of TPC-C in relational database to MongoDB NoSQL data model [12]. In this section, the design of MongoDB has been modeled from TPC-C schema. Mapping of these nine tables (warehouse, customer, district, history, new_order, item, order, order_line and stock) into collection of MongoDB is performed.

5.3 Results and Discussions

System uses NoSQL MongoDB Cloud Data Store for the experiments of the proposed system for scalable workload-driven partitioning system and the developed system is put under hammer in many situations, to prove its authenticity as mentioned in below tests:

5.3.1 Response Time for TPC-C New Order Transactions

Time comparison for TPC-C new order transaction can be depicted in the below shown table 1.

Here, we are comparing the results of new order transaction of TPC-C with the scalable workload-driven partitioning in MongoDB. As observed from results is generates less number of distributed transactions and required low response time.

5.3.2 Response Time for TPC-C Payment Transaction

Time comparison for **TPC-C** payment transaction can be depicted in the below shown Table 2.

Here, we are comparing the results of payment transaction of TPC-C with the scalable workload-driven partitioning in MongoDB. As observed from results is generates less number of distributed transactions and required low response time.

5.3.3 Response Time for TPC-C Delivery Transaction

Time comparison for **TPC-C** Delivery transaction can be depicted in the below shown Table 3.

Here, we are comparing the results of delivery transaction of TPC-C with the scalable workload-driven partitioning in MongoDB. As observed from results is generates less number of distributed transactions and required low response time.

5.3.4 Response Time for TPC-C Stock level Transaction

Time comparison for **TPC-C** stock level transaction can be depicted in the below shown table 4.

Here, we are comparing the results of stock level transaction of TPC-C with the scalable workload-driven partitioning in MongoDB. As observed from results is generates less number of distributed transactions and required low response time.

5.3.5 Response Time for TPC-C Order Status Transaction

Time comparison for **TPC-C** order status transaction can be depicted in the below shown table 5.

Here, we are comparing the results of stock level transaction of TPC-C with the scalable workload-driven partitioning in MongoDB. As observed from results is generates less number of distributed transactions and required low response time.

5.4 Experimental Setup

Some experiments are taken to show the performance of the proposed system on windows machine using java Netbeans as IDE, where 4 machines are considered for experiments which are having core i3 processor with 2GB of Primary memory for the distributed paradigm. Machines are connected to form a distributed environment using CAT 5 cable. MongoDB having version 3.0.2 is used for experiments.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

6. CONCLUSION

Here, scalable workload-driven partitioning in MongoDB is implemented to fulfill the requirements of latest cloud related applications. The solutions by experimentation over MongoDB cloud data store are validated. The industry benchmark TPC-C for assessment of partitioning scheme is utilized. By implementing the concerned scheme using the benchmark TPC-C, it has been observed that scalable workload-driven partitioning in MongoDB reduces the number of distributed transactions and gives lesser response time as compared to TPC-C.

REFERENCES

[1] S. Ahirrao and R. Ingle, "Scalable Transactions in Cloud Data Stores," *in 3rd IEEE International Advance Computing Conference*, India, 2013, pp. 978-1-4673-4529.

[2] S. Phansalkar and Dr. A. Dani, "Transaction Aware Vertical Partitioning Of Database (tavpd) For Responsive Oltp Applications In Cloud Data Stores," *Journal of Theoretical and Applied Information Technology*, Vol. 59 No.1, January 2014.

[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, "Dynamo: Amazon's Highly Available Key-Value store," *proc. of the 21st ACM Symposium on Operating Systems Principles*, 2007, pp. 978-1-59593-591.

[4] S. Das, S. Agarwal, D. Agrawal, and A. ElAbbadi, "ElasTraS: An Elastic, Scalable, and Self Managing Transactional Database for the Cloud," *in Technical Report 2010-04, CS, UCSB*, 2010.

[5] Curino, E. Jones, Y. Zhang, and S. Madden, "Schism: A Workload- Driven Approach to Database Replication and Partitioning," *proc. Of the VLDB Endowment*, Vol. 3, No. 1, 2010.

[6] Bin Liu, Junichi Tatemura, Oliver Po, Wang-Pin Hsiung, Hakan Hacıg¨um¨us, "Automatic Entity-Grouping for OLTP Workloads," *in 30th IEEE International Conference On Data Engineering*, USA, 2014, pp. 978-1-4799-2555.

[7] J. Baker, C. Bond, J. C. Corbett, J. Furman, A. Khorlin, J. Larson, J.-M. Leon, Y. Li, A. Lloyd and V. Yushprakh, "Megastore: Providing Scalable, Highly Available Storage for Interactive Services," *in 5th Biennial Conference on Innovative Data Systems Research*, USA, 2011.

[8] J. J. Levandoski, D. Lomet, M. F. Mokbel and K.K. Zhao, "Deuteronomy: Transaction Support for Cloud Data," *in 5th Biennial Conference on Innovative Data Systems Research*, USA, 2011.

[9] Wei, G. Pierre, and C.-H. Chi. (December 2012). CloudTPS: Scalable Transactions for Web Applications in the Cloud. *IEEE Trans. On Services Computing Vol. 5, N0. 4*. Available: http://www.globule.org/cloudtps.

[10] P. A. Bernstein, I. Cseri, N. Dani, N. Ellis, A.Kalhan, G. Kakivaya, D.B. Lomet, R. Manner, L. Novik and T. Talius, "Adapting Microsoft SQL Server for Cloud Computing," *in 27th International Conference on Data Engineering*, 2011.

[11] D. Agrawal, A. El Abbadi, S. Antony, and S. Das, ".Data Management Challenges in Cloud Computing Infrastructures," *Proc. of the 6th international conference on Databases in symposium on Cloud computing*, 2010, pp. 1-10.

[12] MongoDB. http://www.mongodb.org/.

[13] S. Ahirrao and R. Ingle, "Dynamic workload-aware partitioning in OLTP cloud data stores," *Journal of Theoretical and Applied Information Technology*, Vol. 60 No.1, February 2014.

[14] S. Ahirrao and R. Ingle, "Scalable Transactions in cloud data stores," *Journal of Cloud Computing: Advances, Systems and Applications*, 2015.

[15] http://www.tpc.org/tpcc/

[16] Grolinger K, Higashino WA, Tiwari A, Capretz MAM (2013),"Data Management in Cloud Environments: NoSQL and NewSQL Data Stores," *Journal of Cloud Computing: Advances Systems and Applications*, 2013.

Journal of Theoretical and Applied Information Technology
15th July 2016. Vol.89. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

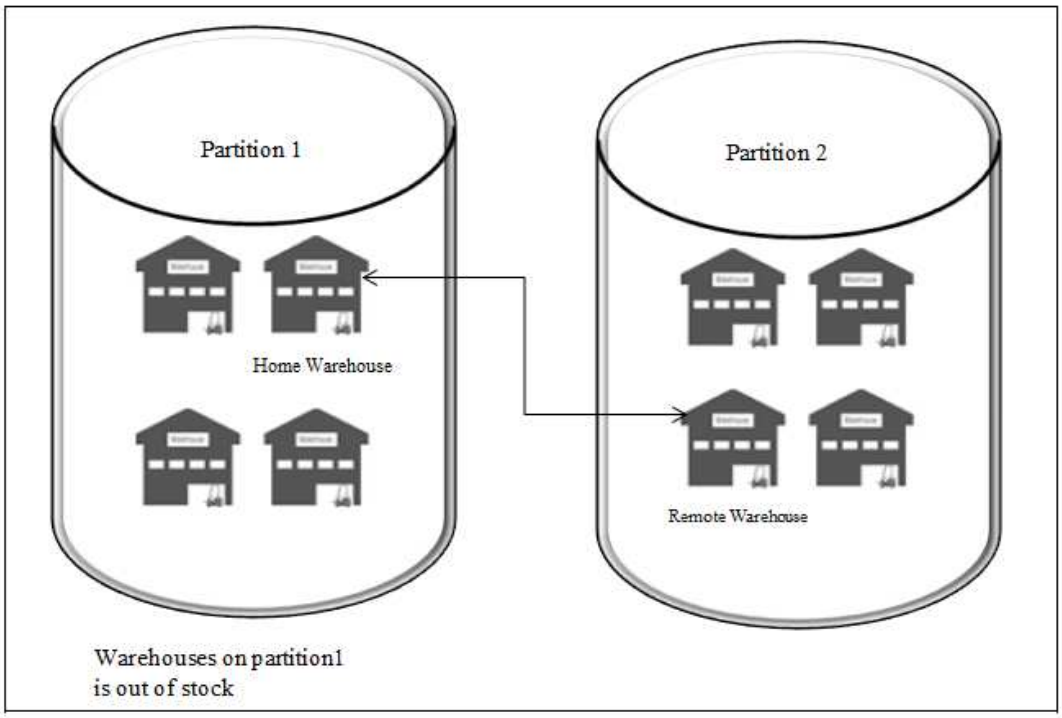

Figure 1: Data Access Patterns

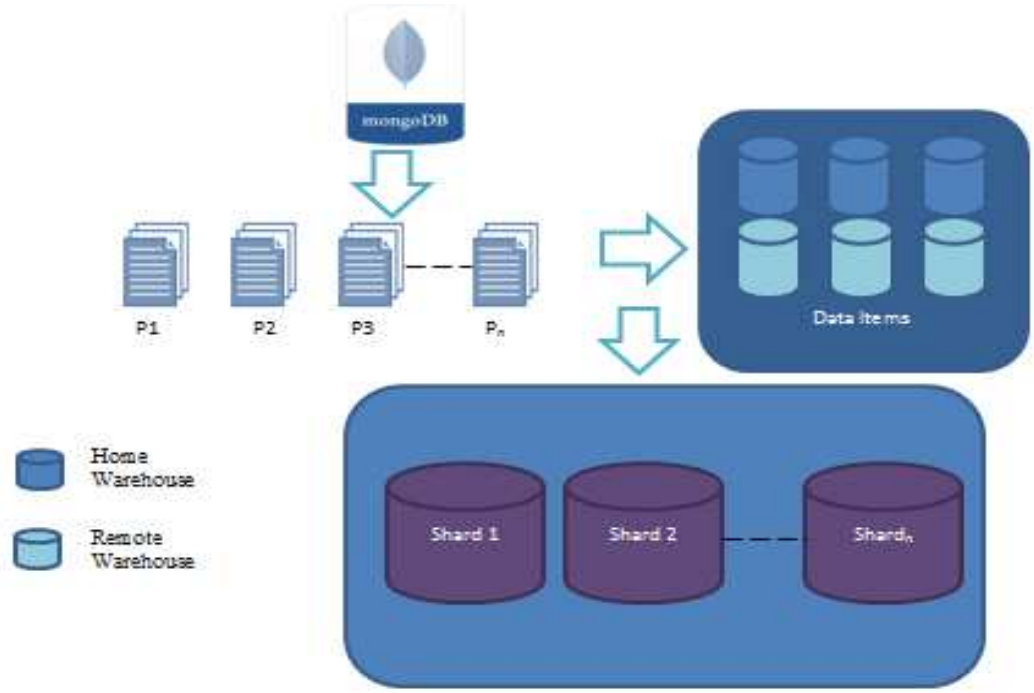

Figure 2: *Design of Scalable Workload Driven Partitioning*

Journal of Theoretical and Applied Information Technology
15th July 2016. Vol.89. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

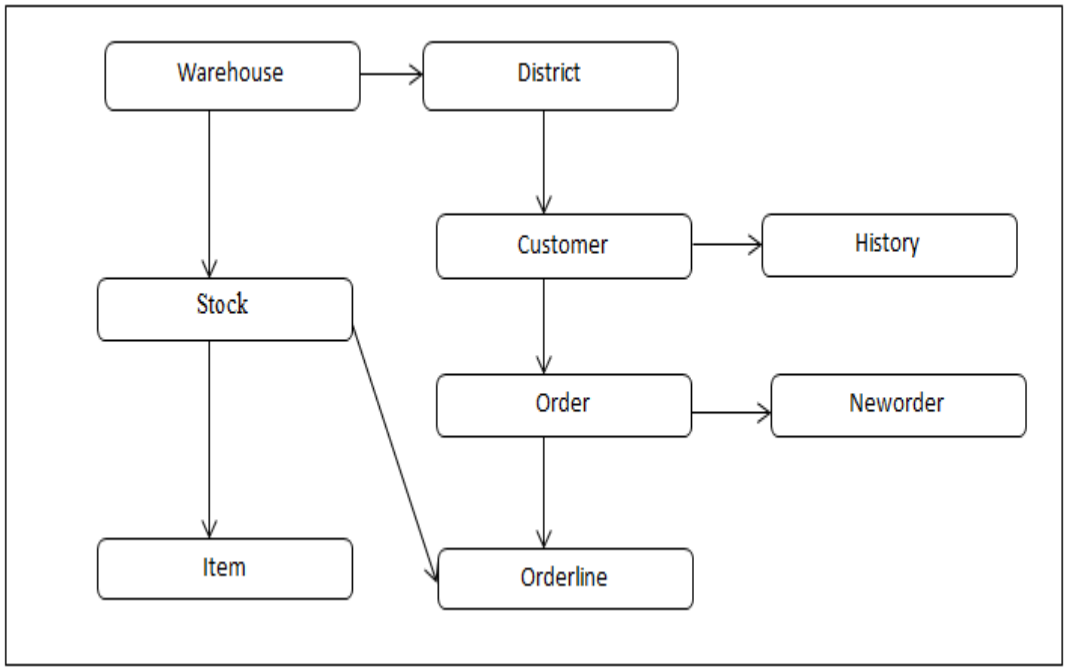

Figure 3: TPC-C Benchmark

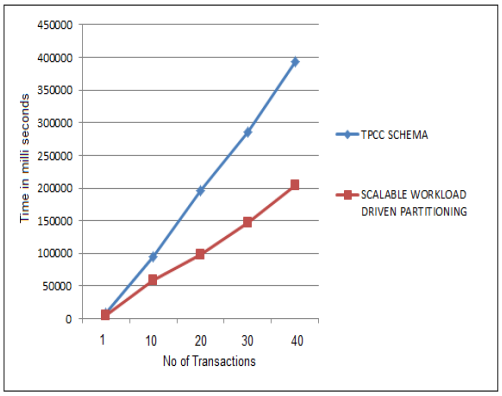

Figure 4: Figure 4: Time Comparison for TPC-C New Order Transaction

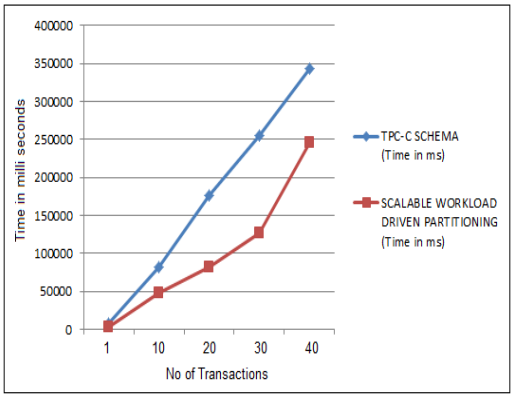

Figure 5: Time Comparison for TPC-C Payment Transaction

Journal of Theoretical and Applied Information Technology
15th July 2016. Vol.89. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

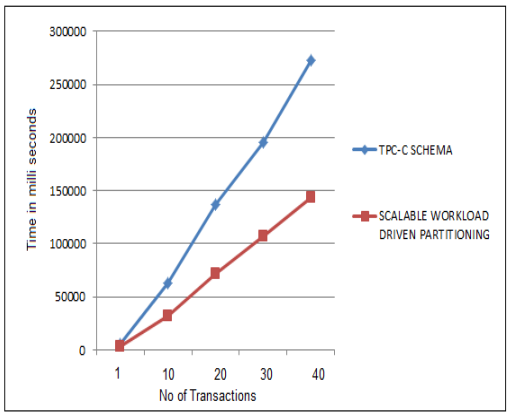

Figure 6: Time Comparison for TPC-C Delivery Transaction

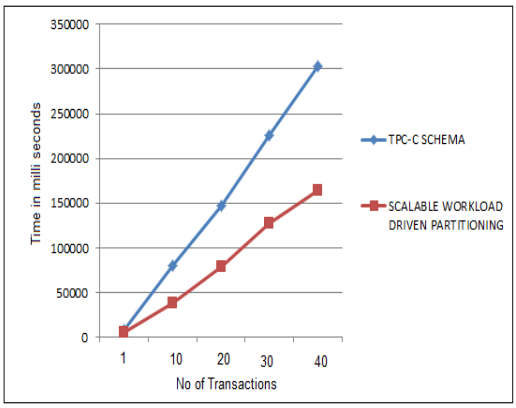

Figure 7: Time Comparison for TPC-C Stock Level Transaction

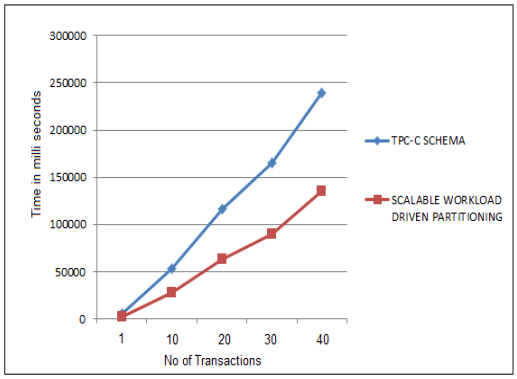

Figure 8: Time Comparison for TPC-C order status Transaction

Table 1: Time Comparison for TPC-C new order Transaction

TPC-C NEW ORDER TRANSACTIONS	TPCC SCHEMA (Time in ms)	SCALABLE WORKLOAD DRIVEN PARTITIONING (Time in ms)
1	9002	4652
10	94789	58234
20	196523	98234
30	285643	147643
40	393076	204523

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 2: Time Comparison for TPC-C payment transaction

TPC-C PAYMENT TRANSACTIONS	TPC-C SCHEMA (Time in ms)	SCALABLE WORKLOAD DRIVEN PARTITIONING (Time in ms)
1	8102	3252
10	81782	48244
20	176523	82234
30	255683	127242
40	343076	245237

Table 3: Time Comparison for TPC-C Delivery Transaction

TPC-C DELIVERY TRANSACTIONS	TPC-C SCHEMA (Time in ms)	SCALABLE WORKLOAD DRIVEN PARTITIONING (Time in ms)
1	6102	2952
10	62782	32244
20	136523	72264
30	195683	107245
40	273076	144113

Table 4: Time Comparison for TPC-C stock level Transaction

TPC-C STOCK TRANSACTIONS	TPC-C SCHEMA (Time in ms)	WORKLOAD DRIVEN PARTITIONING (Time in ms)
1	8102	4752
10	79782	38244
20	146823	79164
30	225683	127982
40	303076	164189

Table 5: Time Comparison for TPC-C Order Status Transaction

TPC-C ORDER STATUS TRANSACTIONS	TPC-C SCHEMA (Time in ms)	SCALABLE WORKLOAD DRIVEN PARTITIONING (Time in ms)
1	5115	2798
10	52981	27678
20	116513	62812
30	165153	89567
40	239841	134873